LBNL-42261
revised

Parallelization of iTOUGH2 Using PVM

Stefan Finsterle

Earth Sciences Division
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720

October 1998

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Geothermal Technologies, of the U.S. Department of Energy, under Contract No. DE-AC03-76SF00098.

ABSTRACT

iTOUGH2 inversions are computationally intensive because the forward problem must be solved many times to evaluate the objective function for different parameter combinations or to numerically calculate sensitivity coefficients. Most of these forward runs are independent from each other and can therefore be performed in parallel. Message passing based on the Parallel Virtual Machine (PVM) system has been implemented into iTOUGH2 to enable parallel processing of iTOUGH2 jobs on a heterogeneous network of Unix workstations. This report describes the PVM system and its implementation into iTOUGH2. Instructions are given for installing PVM, compiling iTOUGH2-PVM for use on a workstation cluster, the preparation of an iTOUGH2 input file under PVM, and the execution of an iTOUGH2-PVM application. Examples are discussed, demonstrating the use of iTOUGH2-PVM.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. iTOUGH2-PVM PARALLELIZATION CONCEPTS 2
   2.1 General Remarks 2
   2.2 Parallelization of the Levenberg-Marquardt Algorithm 3
   2.3 Parallelization of the Gauss-Newton Algorithm 4
   2.4 Parallelization of the Simplex Algorithm 4
   2.5 Parallelization of Grid Search 4
   2.6 Parallelization of First-Order-Second-Moment and Sensitivity Analyses 5
   2.7 Parallelization of Monte Carlo Simulations 5
   2.8 Summary 6

3. INSTALLATION 7
   3.1 Introduction 7
   3.2 Installing PVM 7
   3.3 Installing iTOUGH2-PVM 8

4. USING iTOUGH2-PVM 9
   4.1 iTOUGH2-PVM Command `>>> PVM` 9
   4.2 Running iTOUGH2-PVM 10

5. EXAMPLES 11
   5.1 Overview 11
   5.2 Example 1: Parameter Estimation 12
   5.3 Example 2: Grid Search 16
   5.4 Example 2: Monte Carlo Simulations 21

6. TROUBLESHOOTING 23

ACKNOWLEDGMENT 26

REFERENCES 26

APPENDIX A: SHELL SCRIPT `itough2` 27

APPENDIX B: iTOUGH2-PVM ARCHITECTURE 31

LIST OF FIGURES

Figure 5.2.1. Excerpt from modified iTOUGH2 input file `sam2p3i`. 12
Figure 5.2.2. Screen dump of messages from iTOUGH2-PVM. 12
Figure 5.2.3. Excerpt from iTOUGH2 output file `sam2p3i.out`; spawning of child processes. 13
Figure 5.2.4. Excerpt from iTOUGH2 output file `sam2p3i.out`; summary. 13
Figure 5.2.5. Excerpt from modified iTOUGH2 input file `sam2p3i`; two hosts. . . . 14
Figure 5.2.6. Excerpt from iTOUGH2 output file `sam2p3i.out`, showing information about first Levenberg-Marquardt step. 14
Figure 5.2.7. Excerpt from iTOUGH2 message file `sam2p3i.msg`, showing information about data exchange between processes. 15
Figure 5.3.1. Excerpt from modified iTOUGH2 input file `sam2p2i`. 16
Figure 5.3.2. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing sorted grid search output and load balance. 18
Figure 5.3.3. Excerpt from modified iTOUGH2 input file `sam2p2i`, showing multiple processes being spawned on the same host. 19
Figure 5.3.4. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing number of TOUGH2 runs performed by each process. 19
Figure 5.3.5. Excerpt from modified iTOUGH2 input file `sam2p2i`; unsorted grid search. 20
Figure 5.3.6. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing unsorted grid search output and load balance. 20
Figure 5.4.1. Excerpt from modified iTOUGH2 input file `sam1p6i`. 21
Figure 5.4.2. Excerpt from iTOUGH2 output file `sam1p6i.out`, showing load balance. 22
Figure 6.1. Potential iTOUGH2-PVM error messages. 24
Figure 6.2. Unsuccessful spawning of iTOUGH2-PVM tasks. 24
Figure A.1. Excerpt from shell script file `itough2`. 27
Figure B.1. Simplified iTOUGH2-PVM flow chart. 32

LIST OF TABLES

Table 2.8.1. Summary of Parallelization Concepts 6
Table 5.1.1. Computer Architectures in the Workstation Cluster 11

1. INTRODUCTION

Message passing based on the Parallel Virtual Machine (PVM) system has been implemented into iTOUGH2 to enable parallel processing of iTOUGH2 jobs on a heterogeneous network of Unix workstations. iTOUGH2 is a program for parameter estimation, sensitivity analysis, and uncertainty propagation analysis [*Finsterle*, 1999a,b,c]. iTOUGH2 is based on the TOUGH2 simulator for nonisothermal multiphase flow in porous and fractured media [*Pruess*, 1987, 1991]. PVM [*Geist et al.*, 1994] is a freely available software library that enables a collection of heterogeneous computers to be used as a concurrent computational resource.

iTOUGH2 inversions are computationally intensive because the forward problem—the prediction of state variables using the TOUGH2 simulator—must be solved many times to evaluate the objective function for different parameter combinations or to numerically calculate sensitivity coefficients. Most of these forward runs are independent from each other and can therefore be performed in parallel. iTOUGH2-PVM is based on the parent-child model, in which the parent process initializes the problem, and disseminates specific information—the individual parameter vectors—to a number of child processes. The child processes then perform one full TOUGH2 simulation, returning the results—the elements of the residual vector—back to the parent task for further processing. This type of parallelization on a high level without interaction among the child processes is termed "embarrassingly parallel" [*Geist et al.*, 1994], and has been implemented into iTOUGH2 for improving the efficiency of inversions using the Levenberg-Marquardt, Gauss-Newton, Downhill Simplex, and Grid Search minimization algorithms. Furthermore, the calculation of the coefficients for sensitivity analyses, as well as uncertainty propagation analyses using first-order-second-moment (FOSM) and Monte Carlo simulations, have been parallelized. Minimization based on the method of Simulated Annealing is not parallelized.

This report describes the PVM system and its implementation into iTOUGH2. Instructions are given for installing PVM, compiling iTOUGH2-PVM for use on a workstation cluster, the preparation of an iTOUGH2 input file under PVM, and the execution of an iTOUGH2-PVM application. Examples are discussed, demonstrating the use of iTOUGH2-PVM.

Running an iTOUGH2 job in parallel requires the following steps:

- Installation of PVM on all the potential host machines;
- Compilation of iTOUGH2-PVM and linking to PVM library routines;
- Providing a list of PVM hosts in the iTOUGH2 input file (command `>>> PVM`);
- Running iTOUGH2-PVM using the shell script `itough2` with argument `-pvm`.

We will describe these steps in detail, after discussing the concept of parallelization for each of the iTOUGH2 applications outlined above.

2. iTOUGH2-PVM PARALLELIZATION CONCEPTS

2.1 General Remarks

The main task of iTOUGH2 is to initiate multiple TOUGH2 simulations with different parameter sets, and to analyze each of the corresponding model outputs at selected calibration points. A new, improved parameter set is then proposed following a certain strategy, which is specific to the chosen minimization method. Some of these forward runs are independent from one another, which makes it possible to run them in parallel on separate processors. Since obtaining the forward solutions consumes the bulk of the CPU time used in an iTOUGH2 inversion, with only a few percent of the time spent in the optimization routines, processing individual TOUGH2 runs in parallel has the potential of significantly reducing the turn-around time of an iTOUGH2 inversion.

The degree to which an iTOUGH2 job can be parallelized, and the maximum attainable efficiency depends on the minimization algorithm chosen, the number of parameters to be estimated, and the number of processors available for parallelization. Since PVM enables distributed computing on a heterogeneous network of Unix workstations, the relative speed of the machines in the cluster also affects the efficiency. Again, depending on the method used and the number of processors available, a machine with a slow response time may be the limiting factor, severely hampering the overall performance, or it may merely lead to a slightly suboptimal execution of the job without greatly affecting the efficiency. Note that the response time of a particular machine in the cluster does not solely depend on the speed of its CPU, but also on the machine's workload and the parameter set it happens to receive from the parent process.

Also depending on the iTOUGH2 application, there is a maximum number of processes, m_{procs}, one can reasonably run in parallel. This number is usually identical to the number of parameters to be estimated, n. The actual number of processors available for parallel computing is denoted by n_{procs}. Even if there are more parameters to be estimated than processors available, it may not necessarily be advantageous to engage all available processors, i.e., fewer processors may lead to an equivalent performance because the extra processors would be idle, waiting for another processor to finish its task. Finally, a significantly slower processor in a network has a relatively small impact on the overall performance if the number of processors available is small compared with the maximum number of processors one could potentially use.

While these rules are only approximate and usually difficult to apply because of the unknown workload on a given machine, it is nevertheless necessary for the user to understand which tasks are parallelized in a given iTOUGH2-PVM application. This will be discussed in the following sections.

2.2 Parallelization of the Levenberg-Marquardt Algorithm

The Levenberg-Marquardt minimization algorithm is a gradient-based method that requires evaluating sensitivity coefficients with respect to each parameter to be estimated. In iTOUGH2, the sensitivity coefficients are calculated using the following forward finite difference quotient:

$$J_{ij} = \frac{\partial z_i}{\partial p_j} \approx \frac{z_i(\mathbf{p}; p_j + \delta p_j) - z_i(\mathbf{p})}{\delta p_j} \tag{2.2.1}$$

Here, z_i is the calculated system response at calibration point i, $i = 1,\ldots,m$, and $\mathbf{p}$ is the parameter vector of length n. The evaluation of the Jacobian $\mathbf{J}$ thus requires $n+1$ TOUGH2 simulations: one run is used to obtain the elements $z_i(\mathbf{p})$, followed by n additional runs, each providing one column of the Jacobian matrix. In each run, one of the parameters is perturbed by a small amount δp_j. These n runs with the perturbed parameter sets are independent and are thus parallelized in iTOUGH2-PVM. The maximum number of processors to participate in this calculation is therefore n. The initial forward run is not performed in parallel.

Once the Jacobian is evaluated, the Levenberg-Marquardt algorithm proposes an update vector $\Delta\mathbf{p}$, which depends on the Levenberg parameter λ as follows:

$$\Delta\mathbf{p} = \left(\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{J} + \lambda\mathbf{D}\right)^{-1}\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{r} \tag{2.2.2}$$

Here, $\mathbf{D}$ is an $n \times n$ diagonal matrix with elements $D_{ii} = \left(\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{J}\right)_{ii}$. In the original iTOUGH2 implementation, if the step $\Delta\mathbf{p}$ is successful (i.e., the objective function is reduced), the Levenberg parameter λ is reduced by the Marquardt parameter ν, and a new Jacobian matrix (2.2.1) is evaluated; if the step is not successful (i.e., led to an increase in the objective function), λ is increased by ν, and a new parameter vector $\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \Delta\mathbf{p}$ is calculated using (2.2.2), until a successful step is obtained. Instead of evaluating the objective function in sequence with either increasing or decreasing λ values, a set of n_{procs} forward runs are initiated simultaneously with various λ values, and the simulation yielding the lowest objective function is identified. If this run constitutes a successful step, optimization continues; if it is an unsuccessful step, another n_{procs} runs are performed with $\lambda_i = \lambda_0 \cdot \nu^i$, $i = 1,\ldots,n_{procs}$, where λ_0 is the Levenberg parameter that yielded the lowest value of the objective function in the previous set of runs. The process is repeated until a successful step can be taken or one of the convergence criteria is met.

As discussed above, the Levenberg-Marquardt minimization algorithm runs in parallel for two separate calculations, (1) the evaluation of the Jacobian matrix, and (2) for trying parameter steps with different Levenberg parameters. The latter is equivalent to performing a limited search for the minimum along the line of endpoints of possible Levenberg-Marquardt steps with various values for λ (see Eq. 2.2.2) at each iteration. This may further improve the performance of the algorithm. However, it changes the solution path compared with that

taken by standard, non-parallelized iTOUGH2. Parallelization can be restricted to the evaluation of the Jacobian matrix by using keyword `JACOBIAN` on the command line (see Section 4.1), leading to results which are consistent with the standard iTOUGH2 solution.

The parallel evaluation of the Jacobian has to be completed first, before multiple step vectors can be tested. This means that some processors may be idle until all columns of the Jacobian are evaluated. In other words, the number of processors n_{procs} should be selected such that all processors are busy. For example, if $m_{procs} = n = 8$ and 7 processors of equivalent speed and work load are available, only $n_{procs} = 4$ should be selected to avoid 6 processors being idle for 50% of the time during the calculation of the Jacobian.

2.3 Parallelization of the Gauss-Newton Algorithm

Parallelization of the Gauss-Newton minimization algorithm is similar to that of the Levenberg-Marquardt method described in the previous section. The only difference is that no test runs with varying Levenberg parameters are performed, i.e., only the evaluation of the Jacobian matrix using forward finite differences needs to be parallelized. (Parallelization of centered finite differences available in standard iTOUGH2 requires storing additional, large arrays, and is therefore not supported by iTOUGH2-PVM.)

2.4 Parallelization of the Simplex Algorithm

Only certain calculations performed by the simplex algorithm are suitable for parallelization. They include the evaluation of n vertices of the initial simplex, the $n+1$ simulations performed during overall contraction, and the final calculation of the sensitivity matrix for the error analysis. The initial run as well as function evaluations performed during reflection, expansion, and one-dimensional contraction of the simplex are not parallelized, limiting the overall increase in efficiency attainable.

2.5 Parallelization of Grid Search

The evaluation of the objective function on a large number of points in the parameter space is well suited for parallelization. In iTOUGH2, the output of the grid search method is a sorted list of parameter sets and the corresponding value of the objective function. The sorting requires that results from parallel TOUGH2 simulations are only accepted in the exact same order as they have been submitted. This may affect the efficiency if one of the processors is slower than the others. However, sorting of the iTOUGH2 grid search output could also be done in a post-processing step, in which case a new parameter set is submitted for evaluation as soon as a processor becomes available, increasing efficiency.

By default, the iTOUGH2 output will be sorted. To allow unsorted output with higher efficiency, keyword `UNSORTED` must be added to the `>>> GRID SEARCH` command line

2.6 Parallelization of First-Order-Second-Moment and Sensitivity Analyses

First-order-second-moment (FOSM) uncertainty propagation analyses and simple sensitivity analyses require a single evaluation of the Jacobian matrix (see Equation 2.2.1) using forward finite differences. (Centered finite differences available in standard iTOUGH2 are not supported by iTOUGH2-PVM.) Similar to the parallelization of the first step in the Levenberg-Marquardt algorithm (see Section 2.2), the initial run is performed on the parent processor. The parent processor then broadcasts the result of the base-case run to all available child processors, and up to n additional TOUGH2 runs are performed in parallel, each supplying one column of the Jacobian matrix.

2.7 Parallelization of Monte Carlo Simulations

A large number of Monte Carlo simulations with randomly generated parameter sets can be run in parallel. The parent process performs the first run, broadcasts the results of the base-case run to all child processors, and initiates new runs with random parameter sets whenever one of the child processors has finished its task. The number of usable processors is only limited by the total number of Monte Carlo simulations required, making this type of uncertainty propagation analysis most suitable for parallelization.

2.8 Summary

Table 2.8.1 provides a quick reference to help a user understand the aspects of a calculation that are being parallelized, to select the number of processors to be added to the cluster, and to estimate the expected efficiency.

Table 2.8.1. Summary of Parallelization Concepts

Method	Calculation Parallelized	Calculation Not Parallelized	Comment
Levenberg-Marquardt	Jacobian Testing steps with different λ s	First run	Centered finite differences not supported. $m_{procs} = n$
Gauss-Newton	Jacobian	First run	Centered finite differences not supported. $m_{procs} = n$
Simplex Algorithm	Initial simplex Overall contraction Final Jacobian	First run Reflection Expansion 1D-contraction	$m_{procs} = n$
Grid Search	All runs	-	Use keyword `UNSORTED` for higher efficiency. $m_{procs} = n_{runs}$
Sensitivity Analyses	Jacobian	First run	Centered finite differences not supported. $m_{procs} = n$
First-Order-Second-Moment uncertainty propagation analysis	Jacobian	First run	Centered finite differences not supported. $m_{procs} = n$
Monte Carlo	All runs but first	First run	$m_{procs} = n_{MC}$

3. INSTALLATION

3.1 Introduction

PVM must be installed on all machines of the workstation cluster, and an iTOUGH2-PVM executable must be built on each machine, properly linked to the routines of the PVM library. PVM may already be available on a machine (type `printenv PVM_ROOT` to locate PVM), or must be installed according to the instructions given in Section 3.2. iTOUGH2-PVM must be compiled and linked using command `make pvm` on any machine, following the instructions given in Section 3.3. If the code is redimensioned or changed, it must be recompiled on all machines.

3.2 Installing PVM

This section gives short instructions for obtaining and installing PVM on a Unix workstation. For details, the reader is referred to *Geist et al.* [1994].

PVM can currently be obtained by anonymous ftp to `netlib2.cs.utk.edu`. File `index` in directory `pvm3` describes the files that can be downloaded. The PVM software can also be requested by sending e-mail to `netlib@ornl.gov` with the message: `send index from pvm3`.

After the PVM distribution is unpacked according to the instructions in file `index`, a directory called `pvm3` is created, preferably in the `$HOME` directory. Two environment variables must be defined, most conveniently in the `.cshrc` file (assuming the C-Shell is used). The first variable is `PVM_ROOT`, which is set to the location of the installed `pvm3` directory, for example:

```
setenv PVM_ROOT $HOME/pvm3
```

The second variable is `PVM_ARCH`, which tells PVM the architecture of the host. Valid `PVM_ARCH` names are given in Table 5.1.1 on page 11 and Table 3.1 of *Geist et al.* [1994], or can be automatically determined by appending the contents of file `$PVM_ROOT/lib/cshrc.stub` to file `.cshrc`. The stub must be placed after variables `PATH` and `PVM_ROOT` are defined. Type `source .cshrc` to invoke the changes.

PVM is automatically built by going to directory `$PVM_ROOT` and typing `make`. The makefile will compile and build `pvm`, `pvmd3`, `libpvm3.a`, `libfpvm3.a`, and `libgpvm3.a`, and place them in a subdirectory `$PVM_ROOT/lib/$PVM_ARCH`.

Next, all the hosts one wishes to use must be listed in file `$HOME/.rhosts`. Furthermore, a host file `$HOME/.xpvm_hosts` should be created, listing all the hosts, prepended by an "&". Login name and password are expected to be identical on all hosts. Otherwise, additional host file options must be given as described in Section 3.8 of *Geist et al.* [1994].

PVM is started by typing `pvm`, which should give back a PVM console prompt `pvm>`. Alternatively, type `pvmd &` to start the pvmd3 daemon. The virtual machine could be configured from the console. However, iTOUGH2-PVM will automatically add the hosts defined in the iTOUGH2 input file (see Section 4.1), so the PVM console can be exited immediately by typing `quit`, leaving daemons and PVM jobs running. In order to shut down PVM, type `pvm` again followed by `halt`. Note that PVM needs only be started on the machine where the parent process of iTOUGH2 will be executed. PVM must not be running on the other hosts, i.e., no file `/tmp/pvmd.<uid>` should exist on these hosts.

PVM error messages are printed to the screen and to the log file `/tmp/pvml.<uid>`. Check Section 6 and Section 3.5 of *Geist et al.* [1994] for troubleshooting of common startup problems.

3.3 Installing iTOUGH2-PVM

File `pvm.f` contains the subroutines that link the iTOUGH2 application to the PVM library. To run iTOUGH2-PVM on a single workstation, where PVM is not available or not needed, file `pvmdummy.f` must be used instead of `pvm.f`; it provides dummy subroutines to satisfy all external references.

The iTOUGH2-PVM executable must be built in directory `$HOME/itough2` by typing `make pvm`. The target "`pvm`" makes sure that file `pvm.o` is added to the list of files to be compiled (see variable `OBJPVM` in file `Makefile` for the complete list of object files; customize it if necessary, e.g., add specific modules such as `t2voc.o` or `ifs.o`). Furthermore, the path to the PVM include files and PVM libraries are provided. The two environment variables `PVM_ROOT` and `PVM_ARCH` discussed in Section 3.2 are used to specify the directories that contain the PVM include files and the PVM libraries, respectively, using the `-I` and `-L` options (see variables `CPVM` and `LPVM` in file `Makefile`):

```
-I$(PVM_ROOT)/include
-L$(PVM_ROOT)/lib/$(PVM_ARCH)
```

Since FORTRAN interfaces to PVM are implemented as library stubs that in turn invoke the corresponding C routines, iTOUGH2-PVM must be linked to two archival libraries, namely the FORTRAN stubs library `libfpvm.a` as well as the C library `libpvm3.a`. Additional libraries may be required depending on the machine's architecture. The libraries are linked using the `-l` option. For example, a multiprocessor Sun SPARC station (`PVM_ARCH=SUNMP`), which uses shared memory, needs the following libraries:

```
-lfpvm3 -lpvm3 -lnsl -lsocket -lthread
```

Most architectures require only the first two archival libraries. Customize variable `CPVM` in file `$HOME/itough2/Makefile` accordingly. If successfully built, the resulting iTOUGH2 executable can be used for both standard iTOUGH2 applications or applications that make use of PVM.

4. USING iTOUGH2-PVM

4.1 iTOUGH2-PVM Command >>> PVM

The hosts used for a given iTOUGH2-PVM application are defined in the iTOUGH2 input file using third-level command >>> PVM, which is a subcommand of commands >> OPTIONS and > COMPUTATION. The command syntax is as follows:

Command

```
>>> PVM: nhosts (JACOBIAN/SLEEP: isleep)
HOST1PVM      hostname_1
HOST2PVM      hostname_2
...           ...
HOSTnhostsPVM hostname_nhosts
```

Parent Command

```
>> OPTION
```

Subcommand

-

Description

The number of hosts *nhosts* is provided on the command line, followed by *nhosts* lines containing the keyword *HOSTiPVM (i=1,...,*nhosts*) and the name of the host. The wild card * must be a unique identifier if more than one iTOUGH2-PVM applications are run simultaneously. The name of the host must be identical to that appended to the file name of the iTOUGH2 executable on that specific machine. A host (especially a multiprocessor machine) may be named several times in the list of hosts. The parent process must not be included in the list. However, a child process may be spawned on the parent host. If the Levenberg-Marquardt algorithm is used, parallelization can be restricted to the evaluation of the Jacobian matrix (see Section 2.2) by using keyword JACOBIAN on the command line. The parent process can be suspended for *isleep* seconds each time it checks for incoming residuals (default: *isleep* = 1).

Example

```
> COMPUTATION
  >> OPTION
     >>> use LEVENBERG-MARQUARDT minimization algorithm
     >>> PVM: 5 processors (parallelize JACOBIAN only, SLEEP for : 1 sec)
         HOST1PVM  presto.lbl.gov
         HOST2PVM  hydra.lbl.gov
         HOST3PVM  hydra.lbl.gov
         HOST4PVM  aqua.eth.edu
         HOST5PVM  telos
     <<<
  <<
```

4.2 Running iTOUGH2-PVM

An iTOUGH2-PVM application is started by using argument `-pvm` of the Unix shell script `itough2` (see Appendix A), for example:

```
itough2 -pvm sampvmi sampvm 3 &
```

The process may or may not be executed in the background. Background execution is recommended, so that commands `prista` and `kit` can be submitted from the same window.

If necessary, an iTOUGH2-PVM process should always be killed using command `kit`, i.e., not by typing ctrl-C, which would only kill the `itough2` script, leaving the iTOUGH2-PVM applications running on all hosts.

Argument `-no_delete` can be used to prevent the deletion of the temporary directories on all hosts after completion of a run; argument `-pvm` must precede argument `-no_delete`.

On all hosts, the iTOUGH2-PVM executable must have been built in directory `$HOME/itough2`, i.e., option `-v` cannot be used in combination with `-pvm`.

The `itough2` argument `-pvm` triggers to following actions:

- A standard iTOUGH2 application is set up by creating a temporary directory `$HOME/it2_<pid>` on the parent host. All input files are copied to the temporary directory.
- The iTOUGH2 input file is parsed by the `itough2` script, extracting the number of hosts, the keywords `*HOSTiPVM` (see Section 4.1), and the corresponding hostnames.
- A temporary directory named `it2_*HOSTiPVM` is created on each host.
- The home directory name of the specific host is appended to file `itough2.fil`.
- All files in the parent host's temporary directory are copied to the temporary directory on the child host, including adjusted file `itough2.fil`.
- The filename `itough2_*HOSTiPVM` is linked to the iTOUGH2 executable on each host.
- The parent process is started. It reads the iTOUGH2 input file and spawns the child processes. The parent process controls the application. Program flow is different for the parent process and the child processes (see Appendix B). All child processes are stopped by the parent process after completion of the run.
- Output files from the parent process are copied back from the temporary directory to the working directory.
- The temporary directories on all hosts are deleted unless an error occurred or option `-no_delete` is used.

5. EXAMPLES

5.1 Overview

The examples discussed in the following sections are taken from the collection of iTOUGH2 sample problems described in *Finsterle* [1999c]. They have been slightly modified to allow parallel processing using iTOUGH2-PVM. Focusing on aspects related to the performance of iTOUGH2-PVM, we will not give any interpretation of forward or inverse modeling results.

Example 1 (Section 5.2) is an introductory problem that demonstrates the use of iTOUGH2-PVM for parameter estimation based on the Levenberg-Marquardt algorithm. Two inversions are performed, the first with only the evaluation of the Jacobian being parallelized, and the second with full parallelization. Example 2 (Section 5.3) shows the differences in performance for the grid search method depending on whether the output list is sorted or unsorted. Finally, Example 3 (Section 5.4) discusses parallelization of Monte Carlo simulations on a multiprocessor machine.

Table 5.1.1 contains a list of the Unix workstations used for running the sample problems. Their relative speed as indicated in the last column was measured by running a typical iTOUGH2 application on a single processor, and normalizing the speed to the slowest machine in the cluster. Recall that it is not the CPU time but the turnaround time of a TOUGH2 forward run that determines the effectiveness of a specific workstation in the cluster.

Table 5.1.1. Computer Architectures in the Workstation Cluster

Host	Architecture Operating System	PVM_ARCH	Relative Speed
scully.lbl.gov	DEC Alpha, DEC OSF-1	ALPHA	16.3
yuc.lbl.gov	Silicon Graphics, IRIX	SGI	15.6
hermes.lbl.gov	DEC Alpha, DEC OSF-1	ALPHA	11.0
hydra.lbl.gov	Sun SPARC multiprocessor, Solaris	SUNMP	10.3
presto.lbl.gov	Sun 4, SPARCstation, Solaris	SUN4SOL2	6.9
kungfu.lbl.gov	DEC Alpha, DEC OSF-1	ALPHA	3.9
itelos.lbl.gov	IBM/RS6000, AIX 3.2	RS6K	3.8
killeen.nersc.gov	CRAY multiprocessor, UNICOS	CRAY	2.3
ifs.lbl.gov	Sun 4, SPARCstation, SunOS	SUN4	1.0

5.2 Example 1: Parameter Estimation

The first example consists of running in parallel Problem 2, Part 3 described in *Finsterle* [1999c]. Eight parameters are estimated using the Levenberg-Marquardt algorithm; only the evaluation of the Jacobian matrix will be parallelized. The parent process is run on an IBM/RS6000 (itelos.lbl.gov), and four hosts are added, two DEC Alphas (hermes.lbl.gov and scully.lbl.gov), a Sun 4 SPARCstation (presto.lbl.gov), and a Sun SPARC multiprocessor (hydra.lbl.gov). Block >>> PVM of the iTOUGH2 input file is shown in Figure 5.2.1. Figure 5.2.2 shows a screen dump from itelos.lbl.gov with the command line and messages printed during the execution of iTOUGH2-PVM.

```
> COMPUTATION
  >> OPTION
     >>> PVM : 4 hosts (parallelize JACOBIAN only, don't SLEEP: 0)
         HOST1PVM  hermes.lbl.gov
         HOST2PVM  scully.lbl.gov
         HOST3PVM  hydra.lbl.gov
         HOST4PVM  presto.lbl.gov
     <<<
  <<
<
```

Figure 5.2.1. Excerpt from modified iTOUGH2 input file `sam2p3i`.

```
itough2 -pvm sam2p3i sam2 3 &

++++++++++++++++++
+ iTOUGH2 started: -pvm sam2p3i sam2 3
++++++++++++++++++

PVM: Number of hosts:  4  hosts (parallelize JACOBIAN only)
PVM: Creating temporary directory it2_HOST1PVM on host hermes.lbl.gov.
PVM: Creating temporary directory it2_HOST2PVM on host scully.lbl.gov.
PVM: Creating temporary directory it2_HOST3PVM on host hydra.lbl.gov.
PVM: Creating temporary directory it2_HOST4PVM on host presto.lbl.gov.

PVM: Running iTOUGH2 in parallel.

PVM: Removing temporary directory it2_HOST1PVM on host hermes.lbl.gov
PVM: Removing temporary directory it2_HOST2PVM on host scully.lbl.gov
PVM: Removing temporary directory it2_HOST3PVM on host hydra.lbl.gov
PVM: Removing temporary directory it2_HOST4PVM on host presto.lbl.gov

++++++++++++++++++++
+ iTOUGH2 terminated: -pvm sam2p3i sam2 3
++++++++++++++++++++
```

Figure 5.2.2. Screen dump of messages from iTOUGH2-PVM.

Figure 5.2.3 shows excerpts from the iTOUGH2 output file `sam2p3i.out`. All processes on the four hosts were successfully spawned by the parent process, and a task identifier (TID) was assigned. File `it2_HOSTi/itough2_HOSTiPVM` was successfully linked to the executable in directory `$HOME/itough2` of the respective host. A warning message indicates that the forward instead of the centered finite-difference quotient will be used for calculating the Jacobian.

Convergence is reached after 8 iterations, after a total of 73 TOUGH2 runs have been performed. The work load was almost equally distributed among the four host processors (see Figure 5.2.4), each performing two forward runs per Jacobian evaluation. The parent processor was solving the forward problem 9 times, namely the initial run plus 8 runs to test the proposed Levenberg-Marquardt step. The parent CPU time for this run was 58 sec as compared to 336 sec if the inversion were performed on itelos.lbl.gov without parallelization. Note that the parent processor is the slowest of all machines in the cluster. Solving the inverse problem on scully.lbl.gov without parallelization requires 86 CPU sec.

```
--- PVM ----------------------------------------------------------------
Task     TID Host                        Executable
-------------------------------------------------------------------------
   1  524290 hermes.lbl.gov              it2_HOST1/itough2_HOST1PVM
   2  786434 scully.lbl.gov              it2_HOST2/itough2_HOST2PVM
   3 1048578 hydra.lbl.gov               it2_HOST3/itough2_HOST3PVM
   4 1310722 presto.lbl.gov              it2_HOST4/itough2_HOST4PVM
-------------------------------------------------------------------------
Parent TID                    :  262147
Number of processes spawned   :       4
Parent process suspended for  :       0 sec.
-------------------------------------------------------------------------

*****  WARNING  *****
* Centered Finite Differences not supported by PVM!
*****  WARNING  *****
```

Figure 5.2.3. Excerpt from iTOUGH2 output file `sam2p3i.out`; spawning of child processes.

```
--- PVM ----------------------------------------------------------------
# Runs   TID Host                        Executable
-------------------------------------------------------------------------
  16  524290 hermes.lbl.gov              it2_HOST1/itough2_HOST1PVM
  16  786434 scully.lbl.gov              it2_HOST2/itough2_HOST2PVM
  17 1048578 hydra.lbl.gov               it2_HOST3/itough2_HOST3PVM
  15 1310722 presto.lbl.gov              it2_HOST4/itough2_HOST4PVM
   9  262147 Master                      Suspended for     0 sec.
-------------------------------------------------------------------------
```

Figure 5.2.4. Excerpt from iTOUGH2 output file `sam2p3i.out`; summary.

The problem was slightly modified by reducing the number of hosts to two, and allowing iTOUGH2-PVM to parallelize both the evaluation of the Jacobian and the testing of parameter updates with different Levenberg parameters. The pertinent block in the iTOUGH2 input file is shown in Figure 5.2.5.

```
> COMPUTATION
  >> OPTION
     >>> PVM using : 2 hosts
         HOST1PVM  hermes.lbl.gov
         HOST2PVM  scully.lbl.gov
     <<<
  <<
<
```

Figure 5.2.5. Excerpt from modified iTOUGH2 input file `sam2p3i`; two hosts.

Figure 5.2.6. shows an excerpt from the iTOUGH2 output file `sam2p3i.out` with information about the first iteration. The gradient is calculated from the Jacobian matrix, which is evaluated in parallel on the two child processors. Next, two parameter steps are calculated (Equation 2.2.2) with two values of the Levenberg parameter (λ_1=0.01 and λ_2=0.001). Each step is of different length and orientation, and is checked against constraints such as maximum step size. The two test parameter sets are then evaluated in parallel on the child processors, and the one leading to the smaller value of the objective function (here λ_1) is accepted as the new parameter set.

```
 ------------------------------------------------------------------------------------------------------------
 ITER TOUGH2 OBJ FUNC. MAX. RESID. EQU.  ABS. K GEYS1+8   KLINK GEYS1+8 POROSITY GEYS1+   INIT. 1 TOPB1
                                        INIT. 1 TOPB2    INIT. 1 TOPB3 Leakage Inlet 2 Leakage Inlet 3
 ------------------------------------------------------------------------------------------------------------
>I  0    1  .13217E+07  .26491E+05  122    -.190000E+02     .700000E+01     .150000E-01     .500000E+06
                                             .170000E+07     .300000E+07    -.120000E+02    -.120000E+02
 J  1 Gradient     =    .10910E+08 (forward)
 ----------------------------------------------------------------
 MS   Parameter No. 3: POROSITY GEYS1+  Step  =   -.391743E-01 exceeds max. step size =    -.200000E-01
 MS   Parameter No. 7: Leakage Inlet 2  Step  =    .772640E+01 exceeds max. step size =     .250000E+00
 MS   Parameter No. 8: Leakage Inlet 3  Step  =    .758835E+01 exceeds max. step size =     .250000E+00
 S    Step size    =    .25285E+06     Scaled step size =   .138004E+01  Levenberg parameter =  .10E-01
 BL   Lower bound hit by parameter No. 3: POROSITY GEYS1+  Lower bound  =   .500000E-02
 MS   Parameter No. 3: POROSITY GEYS1+  Step  =   -.652284E-01 exceeds max. step size =    -.200000E-01
 MS   Parameter No. 7: Leakage Inlet 2  Step  =    .831053E+01 exceeds max. step size =     .250000E+00
 MS   Parameter No. 8: Leakage Inlet 3  Step  =    .794512E+01 exceeds max. step size =     .250000E+00
 S    Step size    =    .33789E+06     Scaled step size =   .141143E+01  Levenberg parameter =  .10E-02
 BL   Lower bound hit by parameter No. 3: POROSITY GEYS1+  Lower bound  =   .500000E-02
 PVM Testing with Levenberg Parameter =  .10E-02 on Processor No.  2,  Objective Function =  .13263E+07
 PVM Testing with Levenberg Parameter =  .10E-01 on Processor No.  1,  Objective Function =  .73459E+06
 Minimum objective function obtained with Levenberg parameter:  .10E-01
 PU   Parameter update:                  -.861718E+00     .113737E+00    -.100000E-01    -.172266E+06
                                        -.663148E+05    -.172805E+06     .250000E+00     .250000E+00
>I  1   13  .73828E+06  .14051E+05  139    -.198574E+02     .711374E+01     .500000E-02     .327734E+06
                                             .163369E+07     .282719E+07    -.117500E+02    -.117500E+02
```

Figure 5.2.6. Excerpt from iTOUGH2 output file `sam2p3i.out`, showing information about first Levenberg-Marquardt step.

Message file `sam2p3i.msg` contains some information about the exchange of data between the parent and child processes. An excerpt is shown in Figure 5.2.7. The parent process (TID = 262168) sends the first and second parameter sets along with some iteration parameters to the two child processes. Note that the base-case parameter set was previously evaluated by the parent process, so that the first child process (TID = 524302) receives the parameter set with the first parameter being perturbed, whereas the second child process (TID = 786446) performs a TOUGH2 simulation with the second parameter being perturbed. In this case, the first child process finished its task first, returning the residual vector to the parent process, which immediately sends out a new parameter set (with the third parameter being perturbed) to the first child process. Similar statements are printed on the hosts to report the receiving of parameter sets and sending of residual vectors. These message files could have been retrieved from the hosts' temporary directories if option `-no_delete` were used.

```
--- PVM ---
TID = 262168 sent message No.  1 to TID = 524302 on Fri Sep 11 15:15
Data sent:     NTOUGHC=   1
               NITER  =   0
               IJAC   =   1
               M      = 198
               N      =   8
               X(1..N)=    -.18996E+02     .70000E+01     .15000E-01     .50000E+06
                            .17000E+07     .30000E+07    -.12000E+02    -.12000E+02

--- PVM ---
TID = 262168 sent message No.  2 to TID = 786446 on Fri Sep 11 15:15
Data sent:     NTOUGHC=   2
               NITER  =   0
               IJAC   =   1
               M      = 198
               N      =   8
               X(1..N)=    -.19000E+02     .70043E+01     .15000E-01     .50000E+06
                            .17000E+07     .30000E+07    -.12000E+02    -.12000E+02

--- PVM ---
TID = 262168 received message No.   1 on Fri Sep 11 15:15
Data received: M      = 198
               R(1..4)=    -.74197E+05    -.22276E+04     .16135E+05    -.77206E+04
               F(1..4)=    -.74197E+02    -.22276E+01     .16135E+02    -.77206E+01

--- PVM ---
TID = 262168 sent message No.  3 to TID = 524302 on Fri Sep 11 15:15
Data sent:     NTOUGHC=   3
               NITER  =   0
               IJAC   =   1
               M      = 198
               N      =   8
               X(1..N)=    -.19000E+02     .70000E+01     .15150E-01     .50000E+06
                            .17000E+07     .30000E+07    -.12000E+02    -.12000E+02
```

Figure 5.2.7. Excerpt from iTOUGH2 message file `sam2p3i.msg`, showing information about data exchange between processes.

The results of this inversion are slightly different from those obtained in the previous run because a different solution path was taken as a result of parallelization. Convergence was actually achieved after 7 iterations, and 10 unsuccessful steps (with 20 different values for λ) were taken before the inversion was terminated. These additional unsuccessful steps in fact increased the total number of TOUGH2 simulations from 73 to 96. Nevertheless, the inversion was completed in 44 CPU seconds, shorter than the previous run, simply because all forward runs (except one) were performed on the significantly faster child processors.

Example 1 demonstrates that the performance of iTOUGH2-PVM depends on many factors such as the relative speed of the computers in the cluster, the choice of the parent processor and its load, and the parallelization option selected, which may affect the solution path taken by the Levenberg-Marquardt algorithm.

5.3 Example 2: Grid Search

Evaluating the objective function on a regular grid in the parameter space provides complete information about the topology of the solution. However, the procedure is computationally expensive and becomes prohibitive if the number of parameters is large. In practice, grid search is limited to the analysis of three or fewer parameters.

The individual grid points of the uniformly discretized parameter space can be evaluated in parallel (Section 2.5). However, if the processors in the cluster vary considerably in speed, the performance of parallel processing may deteriorate as will be demonstrated in the first part of this section. Two solutions to the performance problem will also be discussed.

We perform a three-dimensional grid search for Problem 2, Part 2 described in *Finsterle* [1999c]. A range was specified for each of the three parameters defined in block `> PARAMETER`, bounding the parameter space. Each axis in the parameter space is subdivided into 4 intervals, requiring a total of $5 \times 5 \times 5 = 125$ TOUGH2 simulations. The virtual machine consists of the parent processor and three child processors, as shown in Figure 5.3.1. Note that ifs.lbl.gov is about 16 times slower than scully.lbl.gov (see Table 5.1.1)

```
> COMPUTATION

  >> OPTIONS
     >>> GRID SEARCH: 4 4 4 intervals, output SORTED
     >>> PVM: 3
         HOST1PVM scully.lbl.gov
         HOST2PVM hermes.lbl.gov
         HOST3PVM ifs.lbl.gov
     <<<
  <<
```

Figure 5.3.1. Excerpt from modified iTOUGH2 input file `sam2p2i`.

By default, the output will be sorted (see Figure 5.3.2), requiring that the simulation results are accepted by the parent process in exactly the same order as the corresponding parameter sets have been submitted to the child processors. This means that the two faster machines are idle most of the time, waiting for ifs.lbl.gov to complete its run. As shown in Figure 5.3.2, each child processor has received the same number of tasks, independent of their relative speed. While the grid search problem was solved in about one third of the time the slowest processor would have needed, it is obvious that running the task on the fastest machine without parallelization would have given a much better performance.

One solution to this problem is to force iTOUGH2-PVM to submit more parameter sets to the faster processors according to their relative speeds, keeping them equally busy. Figure 5.3.3 shows the corresponding `>>> PVM` block. In order not to overload the faster machines, only 8 processes are started on scully.lbl.gov, 5 on hermes.lbl.gov, and one on ifs.lbl.gov. The result is shown in Figure 5.3.4. As expected, each spawned process carried out the same number of forward runs. However, the total number of TOUGH2 solutions calculated by the fast machine scully.lbl.gov is 72 as compared to 42 in the previous case (see Figure 5.3.2), whereas only 8 runs were performed on the slow ifs.lbl.gov as opposed to 41 before. While improving the performance by about a factor of five, running the problem on scully.lbl.gov alone would still be slightly faster! Parallelization is only advantageous in this case if the number of processes initiated on the three machines were in the ratio of 16:10:1. Note that this example was specially designed to demonstrate a rather extreme case. However, it may reflect the situation encountered on a highly heterogeneous network. It is also important to realize that this poor performance is a result of the sorted grid search algorithm chosen in this example. The other methods discussed in Section 2 exhibit fewer restrictions, i.e., the slowest machine may not be the factor limiting the performance.

This last point is illustrated in the final part of this example, where the output of the grid search is allowed to be unsorted. Adding keyword `UNSORTED` to the line with the command `>>> GRID SEARCH` (see Figure 5.3.5) allows the slowest machine to make its (minor) contribution to the overall task. More importantly, it does not inhibit the performance of the other two hosts. The grid search output shown in Figure 5.3.6 is unsorted. The first parameter set was submitted for evaluation by scully.lbl.gov. However, the result obtained with the second parameter set calculated by hermes.lbl.gov was returned sooner, and the first objective function calculated by ifs.lbl.gov is reported on the tenth line. It becomes obvious that the work load of scully.lbl.gov was relatively high at the time of this simulation; it completed fewer runs than hermes.lbl.gov, and only about 10 times as many as the slow ifs.lbl.gov. This reminds us that it is not the speed of the CPU as tabulated in Table 5.1.1 that determines the overall performance, but the work load of each processor spawned by iTOUGH2-PVM.

The transfer rate of data on the network may also affect the performance especially in these examples, where each TOUGH2 simulation requires only a few CPU seconds. Since only few data are exchanged between the parent process and its children, the impact of the network on the overall performance decreases as the size of the application increases.

```
EVALUATE OBJECTIVE FUNCTION
---------------------------

PARAMETER                       RANGE                SUBDIVISIONS
ABS. K GEYS1+8     -0.20000E+02 <-> -0.19000E+02              4
KLINK GEYS1+8       0.60000E+01 <->  0.70000E+01              4
POROSITY GEYS1+     0.50000E-02 <->  0.10000E+00              4

TOTAL NUMBER OF FUNCTION EVALUATIONS:  125

  ABS. K GEYS1+8   KLINK GEYS1+8 POROSITY GEYS1+  OBJECTIVE FUNC
  -0.2000000E+02   0.6000000E+01   0.5000000E-02   0.2442981E+06
  -0.1975000E+02   0.6000000E+01   0.5000000E-02   0.3665462E+06
  -0.1950000E+02   0.6000000E+01   0.5000000E-02   0.5213457E+06
  -0.1925000E+02   0.6000000E+01   0.5000000E-02   0.7119997E+06
  -0.1900000E+02   0.6000000E+01   0.5000000E-02   0.9369772E+06
  -0.2000000E+02   0.6250000E+01   0.5000000E-02   0.2253501E+06
  -0.1975000E+02   0.6250000E+01   0.5000000E-02   0.3542300E+06
  -0.1950000E+02   0.6250000E+01   0.5000000E-02   0.5261435E+06
  -0.1925000E+02   0.6250000E+01   0.5000000E-02   0.7389479E+06
  -0.1900000E+02   0.6250000E+01   0.5000000E-02   0.9833694E+06
  -0.2000000E+02   0.6500000E+01   0.5000000E-02   0.2105522E+06
  -0.1975000E+02   0.6500000E+01   0.5000000E-02   0.3564799E+06
  -0.1950000E+02   0.6500000E+01   0.5000000E-02   0.5519890E+06
  -0.1925000E+02   0.6500000E+01   0.5000000E-02   0.7863536E+06
  -0.1900000E+02   0.6500000E+01   0.5000000E-02   0.1046639E+07
  -0.2000000E+02   0.6750000E+01   0.5000000E-02   0.2110985E+06
  -0.1975000E+02   0.6750000E+01   0.5000000E-02   0.3814090E+06
  -0.1950000E+02   0.6750000E+01   0.5000000E-02   0.5997934E+06
  -0.1925000E+02   0.6750000E+01   0.5000000E-02   0.8511938E+06
  -0.1900000E+02   0.6750000E+01   0.5000000E-02   0.1122943E+07
  -0.2000000E+02   0.7000000E+01   0.5000000E-02   0.2352778E+06
  -0.1975000E+02   0.7000000E+01   0.5000000E-02   0.4292179E+06
  -0.1950000E+02   0.7000000E+01   0.5000000E-02   0.6653426E+06
  -0.1925000E+02   0.7000000E+01   0.5000000E-02   0.9287278E+06
  -0.1900000E+02   0.7000000E+01   0.5000000E-02   0.1208063E+07
  -0.2000000E+02   0.6000000E+01   0.2875000E-01   0.2430781E+06
  -0.1975000E+02   0.6000000E+01   0.2875000E-01   0.3744522E+06
   ...             ...             ...             ...
  -0.1925000E+02   0.6750000E+01   0.1000000E+00   0.1188086E+07
  -0.1900000E+02   0.6750000E+01   0.1000000E+00   0.1487237E+07
  -0.2000000E+02   0.7000000E+01   0.1000000E+00   0.4704075E+06
  -0.1975000E+02   0.7000000E+01   0.1000000E+00   0.7061561E+06
  -0.1950000E+02   0.7000000E+01   0.1000000E+00   0.9799752E+06
  -0.1925000E+02   0.7000000E+01   0.1000000E+00   0.1276334E+07
  -0.1900000E+02   0.7000000E+01   0.1000000E+00   0.1581797E+07

Terminated normally.

--- PVM ----------------------------------------------------------------
# Runs   TID Host                      Executable
-------------------------------------------------------------------------
  42  524297 scully.lbl.gov            it2_HOST1PVM/itough2_HOST1PVM
  42 1048587 hermes.lbl.gov            it2_HOST2PVM/itough2_HOST2PVM
  41  786443 ifs.lbl.gov               it2_HOST3PVM/itough2_HOST3PVM
   0  262159 Master                    Suspended for     0 sec.
-------------------------------------------------------------------------
```

Figure 5.3.2. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing sorted grid search output and load balance.

```
> COMPUTATION

  >> OPTIONS
     >>> GRID SEARCH: 4 4 4 intervals, output SORTED
     >>> PVM: 14
         HOST1PVM  scully.lbl.gov
         HOST2PVM  scully.lbl.gov
         HOST3PVM  scully.lbl.gov
         HOST4PVM  scully.lbl.gov
         HOST5PVM  scully.lbl.gov
         HOST6PVM  scully.lbl.gov
         HOST7PVM  scully.lbl.gov
         HOST8PVM  scully.lbl.gov
         HOST9PVM  hermes.lbl.gov
         HOST10PVM hermes.lbl.gov
         HOST11PVM hermes.lbl.gov
         HOST12PVM hermes.lbl.gov
         HOST13PVM hermes.lbl.gov
         HOST14PVM ifs.lbl.gov
     <<<
  <<
```

Figure 5.3.3. Excerpt from modified iTOUGH2 input file `sam2p2i`, showing multiple processes being spawned on the same host.

```
 --- PVM ----------------------------------------------------------------
 # Runs   TID Host                       Executable
 -------------------------------------------------------------------------
    9  524298 scully.lbl.gov             it2_HOST1PVM/itough2_HOST1PVM
    9  524299 scully.lbl.gov             it2_HOST2PVM/itough2_HOST2PVM
    9  524300 scully.lbl.gov             it2_HOST3PVM/itough2_HOST3PVM
    9  524301 scully.lbl.gov             it2_HOST4PVM/itough2_HOST4PVM
    9  524302 scully.lbl.gov             it2_HOST5PVM/itough2_HOST5PVM
    9  524303 scully.lbl.gov             it2_HOST6PVM/itough2_HOST6PVM
    9  524304 scully.lbl.gov             it2_HOST7PVM/itough2_HOST7PVM
    9  524305 scully.lbl.gov             it2_HOST8PVM/itough2_HOST8PVM
    9 1048588 hermes.lbl.gov            it2_HOST9PVM/itough2_HOST9PVM
    9 1048589 hermes.lbl.gov            it2_HOST10PVM/itough2_HOST10PVM
    9 1048590 hermes.lbl.gov            it2_HOST11PVM/itough2_HOST11PVM
    9 1048591 hermes.lbl.gov            it2_HOST12PVM/itough2_HOST12PVM
    9 1048592 hermes.lbl.gov            it2_HOST13PVM/itough2_HOST13PVM
    8  786444 ifs.lbl.gov               it2_HOST14PVM/itough2_HOST14PVM
    0  262160 Master                    Suspended for      0 sec.
 -------------------------------------------------------------------------
```

Figure 5.3.4. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing number of TOUGH2 runs performed by each process.

```
> COMPUTATION

  >> OPTIONS
     >>> GRID SEARCH: 4 4 4 intervals, output UNSORTED
     >>> PVM: 3
         HOST1PVM scully.lbl.gov
         HOST2PVM hermes.lbl.gov
         HOST3PVM ifs.lbl.gov
     <<<
  <<
```

Figure 5.3.5. Excerpt from modified iTOUGH2 input file `sam2p2i`; unsorted grid search.

```
EVALUATE OBJECTIVE FUNCTION
---------------------------

PARAMETER                      RANGE               SUBDIVISIONS
ABS. K GEYS1+8     -0.20000E+02 <-> -0.19000E+02              4
KLINK GEYS1+8       0.60000E+01 <->  0.70000E+01              4
POROSITY GEYS1+     0.50000E-02 <->  0.10000E+00              4

TOTAL NUMBER OF FUNCTION EVALUATIONS:  125

  ABS. K GEYS1+8   KLINK GEYS1+8 POROSITY GEYS1+  OBJECTIVE FUNC
  -0.1975000E+02   0.6000000E+01   0.5000000E-02   0.3665462E+06
  -0.2000000E+02   0.6000000E+01   0.5000000E-02   0.2442981E+06
  -0.1925000E+02   0.6000000E+01   0.5000000E-02   0.7119997E+06
  -0.1900000E+02   0.6000000E+01   0.5000000E-02   0.9369772E+06
  -0.2000000E+02   0.6250000E+01   0.5000000E-02   0.2253501E+06
  -0.1975000E+02   0.6250000E+01   0.5000000E-02   0.3542300E+06
  -0.1925000E+02   0.6250000E+01   0.5000000E-02   0.7389479E+06
  -0.1900000E+02   0.6250000E+01   0.5000000E-02   0.9833694E+06
  -0.2000000E+02   0.6500000E+01   0.5000000E-02   0.2105522E+06
  -0.1950000E+02   0.6250000E+01   0.5000000E-02   0.5261435E+06
  -0.1975000E+02   0.6500000E+01   0.5000000E-02   0.3564799E+06
  -0.1950000E+02   0.6500000E+01   0.5000000E-02   0.5519890E+06
  -0.1925000E+02   0.6500000E+01   0.5000000E-02   0.7863536E+06
  -0.1900000E+02   0.6500000E+01   0.5000000E-02   0.1046639E+07
  -0.2000000E+02   0.6750000E+01   0.5000000E-02   0.2110985E+06
  -0.1975000E+02   0.6750000E+01   0.5000000E-02   0.3814090E+06
  -0.1950000E+02   0.6000000E+01   0.5000000E-02   0.5213457E+06
  -0.1950000E+02   0.6750000E+01   0.5000000E-02   0.5997934E+06
  -0.1925000E+02   0.6750000E+01   0.5000000E-02   0.8511938E+06
   ...             ...             ...             ...

--- PVM ----------------------------------------------------------------------
# Runs   TID Host                     Executable
-------------------------------------------------------------------------------
  57  524296 scully.lbl.gov           it2_HOST1PVM/itough2_HOST1PVM
  62 1048586 hermes.lbl.gov           it2_HOST2PVM/itough2_HOST2PVM
   6  786442 ifs.lbl.gov              it2_HOST3PVM/itough2_HOST3PVM
   0  262158 Master                   Suspended for      0 sec.
-------------------------------------------------------------------------------
```

Figure 5.3.6. Excerpt from iTOUGH2 output file `sam2p2i.out`, showing unsorted grid search output and load balance.

5.4 Example 3: Monte Carlo Simulations

iTOUGH2-PVM can also be installed on a multiprocessor machine such as the Cray J90 killeen.nersc.gov. No effort has been made to optimize the performance of iTOUGH2 on this vector machine, or to take advantage of its built-in parallelization capabilities. Problem 1 Part 6 described in *Finsterle* [1999c] was selected to demonstrate the use of iTOUGH2-PVM for Monte Carlo simulations on a single multiprocessor machine. Figure 5.4.1 shows block `> PVM` from the modified iTOUGH2 input file `sam1p6i`. Only 8 of 32 available processors were used because the priority is automatically reduced if too many processors are occupied by the same user. The same host name is listed eight times. Recall that eight different temporary subdirectories will be created on the host (see Section 4.2), avoiding potential file conflicts. Note that additional processors on different machines could have been added.

```
> COMPUTATION
  >> OPTION
     >>> Use: 8 PVM processors on Cray J90, parent SPEEP for : 1 second
         HOST1PVM  killeen.nersc.gov
         HOST2PVM  killeen.nersc.gov
         HOST3PVM  killeen.nersc.gov
         HOST4PVM  killeen.nersc.gov
         HOST5PVM  killeen.nersc.gov
         HOST6PVM  killeen.nersc.gov
         HOST7PVM  killeen.nersc.gov
         HOST8PVM  killeen.nersc.gov
     <<<
  <<
```

Figure 5.4.1. Excerpt from modified iTOUGH2 input file `sam1p6i`.

The initial run with the mean parameter set is performed by the parent process, which is just another process running on killeen.nersc.gov. After completion, the resulting mean system behavior, which is information needed by the child processes, is broadcast to all eight of them, and the first eight forward runs with random parameter sets are initiated. As soon as a run is completed, the next random parameter set is calculated and submitted to the free processor, i.e., all eight child processes are constantly and simultaneously performing simulations. As shown in Figure 5.4.2, the 100 requested Monte Carlo simulations are almost uniformly distributed over the eight child processes; the parent process performed only one (the first) forward run, sent and received data, and conducted the final statistical analysis. The task was completed in 16% of the time needed if the analysis were performed using only one processor.

Note that Monte Carlo simulations and unsorted grid search do not require the good load balance seen in this example. Good load balance is crucial for sorted grid search (see Section 5.3, Part 1), and is important for parallelization of the Levenberg-Marquardt algorithm,

especially if the number of parallel processes approaches the number of parameters to be estimated.

```
--- PVM ----------------------------------------------------------------------------
# Runs   TID Host                      Executable
-------------------------------------------------------------------------------------
  13  262147 killeen                   it2_HOST1PVM/itough2_HOST1PVM
  13  262148 killeen                   it2_HOST2PVM/itough2_HOST2PVM
  12  262149 killeen                   it2_HOST3PVM/itough2_HOST3PVM
  13  262150 killeen                   it2_HOST4PVM/itough2_HOST4PVM
  12  262151 killeen                   it2_HOST5PVM/itough2_HOST5PVM
  13  262152 killeen                   it2_HOST6PVM/itough2_HOST6PVM
  12  262153 killeen                   it2_HOST7PVM/itough2_HOST7PVM
  11  262154 killeen                   it2_HOST8PVM/itough2_HOST8PVM
   1  262146 Master                    Suspended for    145 sec.
-------------------------------------------------------------------------------------
```

Figure 5.4.2. Excerpt from iTOUGH2 output file `sam1p6i.out`, showing load balance.

In this and most other examples, the parent process performs only one forward run. After the initial run is completed, it spends most of its time in a loop, continuously checking whether a residual vector from one of the child processes has arrived. When a residual vector is received, the parent process performs a few minor calculations, prepares the next parameter set, sends it out, and resumes its waiting position. Since it is not waiting for a particular child process or a particular message ID, the parent process is constantly checking for messages, thus using CPU time. With keyword `SLEEP` on the command line (see Section 4.1), the execution of the parent process can be suspended for a certain amount of time whenever it rechecks all the child processes for messages. This saves CPU time on the parent process, which could be utilized, for example, by another child process spawned on the parent host itself. A sleeping time of one second is reasonable. For most large iTOUGH2-PVM applications, it is recommended that keyword `SLEEP` be used, providing the possibility of running a child process on the parent machine.

6. TROUBLESHOOTING

Successful use of iTOUGH2-PVM requires experience with running standard iTOUGH2 applications, good understanding of the concepts described in Section 2 of this report, and some knowledge of the Unix operating system. It is also important to understand the `itough2` shell script as outlined in Section 4.2 and Appendix A

This section attempts to summarize common problems encountered by users when installing and running iTOUGH2-PVM, and offers some guidelines to fix them. It also explains some of the error messages potentially generated by the code.

Installing PVM

Follow the instructions in Section 3.2, and consult *Geist et al.* [1994] for additional troubleshooting. Make sure that the environment variables `PVM_ROOT` and `PVM_ARCH` are set correctly. A FORTRAN77 and C compiler are required.

Starting PVM

Start PVM by either entering the PVM console or starting the pvmd daemon using, respectively, the scripts `pvm` or `pvmd`; the scripts are located in directory `$PVM_ROOT/lib`, which must be added to the search path. The pvmd writes error messages to a log file named `/tmp/pvml.<uid>`, where `<uid>` is a numeric user identifier. PVM cannot be started if the socket address file `/tmp/pvmd.<uid>` exists from a previous run that was killed with an uncatchable signal. This file must be removed before another pvmd will start.

Installing iTOUGH2-PVM

To install iTOUGH2-PVM, type `make pvm` (see Section 3.3). If include file `fpvm3.h` cannot be found during compilation, check variable `CPVM` in file `$HOME/itough2/Makefile`, or copy file `fpvm3.h` from directory `$PVM_ROOT/include` to directory `$HOME/itough2`. If the PVM library routines are not found during linking, check variable `LPVM` in file `$HOME/itough2/Makefile`. Make sure that the environment variables `PVM_ROOT` and `PVM_ARCH` are set correctly.

Running iTOUGH2-PVM

iTOUGH2-PVM is started using the script `itough2` with argument `-pvm`. Messages similar to those shown in Figure 5.2.2 should be printed to the screen.

If error messages from commands `rcp` or `rsh` appear on the screen, check for correct hostnames and keywords `HOSTiPVM` in the iTOUGH2 input file, as well as for permissions on the corresponding host. You must have the same login name and password on all hosts. The host should be registered in files `/etc/hosts` and `$HOME/.rhosts`.

After completion of the iTOUGH2-PVM run, check for error messages in the iTOUGH2 output file, the iTOUGH2 message file, or file `/tmp/pvml.<uid>`. You may use

command options `-pvm -no_delete` to prevent removal of the temporary directories on all the hosts. Check for additional error messages in the corresponding files on each host.

The set of error messages shown in Figure 6.1 are related to improper installation of PVM or iTOUGH2-PVM on the parent host, as well as wrong command usage; the messages and remedies are self-explanatory.

The result of initiating child processes is reported in the iTOUGH2 output file (see Figure 5.3.2). An example of unsuccessfully spawned tasks is shown in Figure 6.2.

```
*****  ERROR  *****
* No PVM routines from file pvm.f found.  Recompile; type 'make pvm'.
*****  ERROR  *****

*****  ERROR  *****
* pvmd not responding!
* Start PVM before running iTOUGH2.  Type 'pvm' followed by 'quit'.
*****  ERROR  *****

*****  ERROR *****
* iTOUGH2 would run locally on a single processor.
* Use argument '-pvm' on itough2 command line.
* Check for correct installation of PVM and iTOUGH2-PVM on all hosts.
*****  ERROR  *****

*****  ERROR *****
* PVM: Task TID  1310724 stopped.
* All PVM tasks stopped!
*****  ERROR  *****
```

Figure 6.1. Potential iTOUGH2-PVM error messages.

```
--- PVM ----------------------------------------------------------------
Task     TID Host                       Executable
-------------------------------------------------------------------------
ERROR     -6 kungfu.lbl.gov             it2_HOST1PVM/itough2_HOST1PVM
   1  524292 kungfu                     it2_HOST2PVM/itough2_HOST2PVM
   2  786436 scully.lbl.gov             it2_HOST3PVM/itough2_HOST3PVM
ERROR     -6 hermes.lbl.gov             it2_HOST4PVM/itough2_HOST4PVM
ERROR     -7 itelos                     it2_HOST5PVM/itough2_HOST5PVM
-------------------------------------------------------------------------
Parent TID                    :  262150
Number of processes spawned   :       2
-------------------------------------------------------------------------
```

Figure 6.2. Unsuccessful spawning of iTOUGH2-PVM tasks.

There are various reasons for an erroneous initiation of child processes. The parent pvmd may fail to add a host and start the child pvmd if:

- PVM is not properly installed on that host (check installation of PVM);
- The parent pvmd cannot resolve the host name to an IP address (check files `/etc/hosts` and `$HOME/.rhosts`);
- A daemon pvmd is already running on the host (stop PVM on all hosts except the parent host by typing `pvm` followed by `halt`).
- The pvmd executable and shell script `pvmd` is not installed in the correct location (set environment variable `PVM_DPATH` on the parent host to `pvm3/lib/pvmd`);
- Something is printed in the `.cshrc` or equivalent script file (move printing statement to file `.login`; see also Section 9.2.5 of *Geist et al.* [1994]).

Other reasons for an error message as shown in Figure 6.2 are related to iTOUGH2. Make sure that the programs `$HOME/itough2/itough2_<eos>.<hostname>` exist and are executable on all hosts. Here, `<eos>` is the number of the equation-of-state module used, and `<hostname>` is the name of the host as printed when typing `hostname`, which may or may not include the domainname. For example, since the domainname is not included when typing hostname on host kungfu.lbl.gov, the executable was not found if kungfu.lbl.gov was given as the hostname; the task was successfully spawned when using hostname kungfu.

Note that if any of the child processes is stopped during an application for any reason, iTOUGH2-PVM terminates (see last error message in Figure 6.1).

ACKNOWLEDGMENT

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Geothermal Technologies, of the U.S. Department of Energy, under Contract No. DE-AC03-76SF00098. I would like to thank R. Hinkins for introducing me to parallel computing. The review comments by C. Doughty, C. Oldenburg and A. Mishra are gratefully acknowledged. Many thanks to D. Hawkes for his editorial comments.

REFERENCES

Finsterle, S., *iTOUGH2 User's Guide*, Report LBNL-40400, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1999a.

Finsterle, S., *iTOUGH2 Command Reference*, Report LBNL-40401, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1999b.

Finsterle, S., *iTOUGH2 Sample Problems*, Report LBNL-40402, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1999c.

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.
(This book can currently be viewed over the Internet at the following URL:
`http://www.netlib.org/pvm3/book/pvm-book.html`
A PostScript file can be retrieved from the anonymous ftp server
`netlib2.cs.utk.edu`, directory `pvm3/book`, file `pvm-book.ps`)

Pruess, K., *TOUGH User's Guide*, Report NUREG/CR-4645, Nuclear Regulatory Commission (also Report LBL-20700, Lawrence Berkeley Laboratory, Berkeley, Calif.), 1987.

Pruess, K., *TOUGH2—A General-Purpose Numerical Simulator for Multiphase Fluid and Heat Flow*, Report LBL-29400, Lawrence Berkeley Laboratory, Berkeley, Calif., 1991.

APPENDIX A: SHELL SCRIPT `ITOUGH2`

Figure A.1 shows an excerpt from the Unix shell script file `itough2`, which is used to start both standard iTOUGH2 applications as well as iTOUGH2-PVM. Type `itough2` without any arguments to obtain the command usage.

The `itough2` script and the iTOUGH2 FORTRAN77 code are interrelated. Corrupting either one may prevent iTOUGH2-PVM from running. The steps performed by the `itough2` script are described in general terms in Section 4.2; see also comments in Figure A.1.

```
#! /bin/sh
#
##############################################################################
# Shell script to run iTOUGH2 (Finsterle, June 1998)                         #
#                                                                            #
# Copy this file to your $HOME/bin directory.                                #
# Set variable prog_dir (see line 22).                                       #
# Make sure that directory $HOME/bin is in your search path.                 #
# Type "chmod a+x itough2" to make itough2 an executable command.           #
#                                                                            #
# Syntax:  itough2 [options] inv_file dir_file ieos                          #
#                                                                            #
#          inv_file  = iTOUGH2 input file                                    #
#          dir_file  = TOUGH2 input file                                     #
#          ieos      = Number of EOS module being used                       #
#                                                                            #
#          Options:  (see below)                                             #
##############################################################################
#
# Provide here the path to the itough2 executable
#
  prog_dir=$HOME/itough2
# Provide here the path to the main temporary directory
#
  tmp_dir=$HOME
##############################################################################
#
# At least three arguments must be given
#
arguments=$*
echo " "
echo ++++++++++++++++++
echo + iTOUGH2 started: $arguments
echo ++++++++++++++++++
if test $# -lt 3
then                                                    (prints command usage)
     exit 1
fi
...
 (initializes variables; checks command arguments; sets pvm=yes for option -pvm)
...
```

Figure A.1. Excerpt from shell script file `itough2`.

```
program=$progdir/itough2_$arg3.`hostname`
if test ! -s "$program"
then                                                    (prints error message)
  exit 1
fi
tmp_dir=$tmp_dir/it2\_$$
datum=`date`
ori_dir=`pwd`
#
mkdir $tmp_dir                          create temporary directory on parent host
cd    $tmp_dir
tmp_dir=`pwd`                                    > $inv_fil.msg 2>&1
inv_fil=`echo $arg1|awk -F. '{ print $1 }'`
dir_fil=`echo $arg2|awk -F. '{ print $1 }'`
if test $inv_out = $default
then inv_out=$inv_fil.out
fi
if test $dir_out = $default
then dir_out=$dir_fil.out
fi
if test $sav_out = $default
then sav_out=$dir_fil.sav
fi
...
                                                  (creates file  itough2.msg)
...
#                                  write input file names into file  itough.fil
#
echo $arg1                                         > itough2.fil
echo $arg2                                        >> itough2.fil
echo $ori_dir                                     >> itough2.fil
echo $datum                                       >> itough2.fil
echo $arguments                                   >> itough2.fil
echo $0                                           >> itough2.fil
echo $tmp_dir                                     >> itough2.fil
#
#                   copy iTOUGH2 and TOUGH2 input files to temporary directory
#
...
cp  $ori_dir/$arg1 .
cp  $ori_dir/$arg2 .
...
                                (copies additional input files to temporary directory)
...
if test $run = yes
then                                  copy potential data files to temporary directory
   cd $ori_dir
   cp `grep -i FILE $inv_fil | awk -F: '{print $2}' \\
      | awk '{print $1}'` $tmp_dir >> $inv_fil.msg  2>&1
   cd $tmp_dir
fi
...
```

Figure A.1. (cont.) Excerpt from shell script file `itough2`.

```
#
# PVM
#
if test $pvm = yes -a $run = yes
then
#
#                                          parse input file to find number of hosts
#
nprocs=`grep ">>>" $inv_fil | grep PVM | awk -F: '{print $2}'`
ip=0
echo " "
echo "PVM: Number of hosts: $nprocs"
cp $tmp_dir/itough2.fil $tmp_dir/itough2.dum
while test "$ip" -lt "$nprocs"
do                                    parse input file to find hosts and directories
  ip=`expr $ip + 1`
  hostpvm="HOST"$ip"PVM"
  remote_dir=it2_`grep $hostpvm $inv_fil | awk '{print $1}'`
  host=`grep -i $hostpvm $inv_fil | awk '{print $2}'`
  echo "PVM: Creating temporary directory $remote_dir on host $host."
  if test $host = `hostname`     parent host and child host are identical; use  cp
  then
    mkdir $HOME/$remote_dir                          create temporary directory
    echo  $HOME >> $tmp_dir/itough2.fil               add home directory name
    cp    $tmp_dir/itough2.fil $HOME
    cp    $tmp_dir/* $HOME/$remote_dir     copy all files to temporary directory
    ln -f $program   $HOME/$remote_dir/itough2_$hostpvm          create link
    cp    $tmp_dir/itough2.dum $tmp_dir/itough2.fil
  else                        parent host and child host are not identical, use  rcp
    rsh -n $host mkdir $remote_dir                   create temporary directory
    remote_home=`rsh -n $host pwd`  add home directory name to  itough2.fil
    echo   $remote_home >> $tmp_dir/itough2.fil
    rcp    $tmp_dir/itough2.fil $host:)
    rcp    $tmp_dir/* $host:$remote_dir                  copy all files to host
    rsh -n $host ln -f itough2/itough2_$arg3.$host \\
           $remote_dir/itough2_$hostpvm              create link to executable
    cp     $tmp_dir/itough2.dum $tmp_dir/itough2.fil
  fi
done
mv $tmp_dir/itough2.dum $tmp_dir/itough2.fil
echo " "
echo "PVM: Running iTOUGH2 in parallel."
fi
#
if test $run = yes
then $program                                      start iTOUGH2 on parent host
#
...
        (copies output files from local directory on parent host to working directory)
...
```

Figure A.1. (cont.) Excerpt from shell script file `itough2`.

```
...
                                                    (removes temporary files)
...
if test $delete = yes
then
  nprocs=`grep ">>>" $inv_fil | grep PVM | awk -F: '{print $2}'`
  ip=0
  if test $pvm = yes
  then
  while test "$ip" -lt "$nprocs"
  do
     ip=`expr $ip + 1`
     hostpvm="HOST"$ip"PVM"
     remote_dir=it2_`grep $hostpvm $inv_fil | awk '{print $1}'`
     host=`grep -i $hostpvm $inv_fil | awk '{print $2}'`
     if test $host = `hostname`  parent host and child host are identical; use  rm
     then
       echo "PVM: Removing temporary directory $remote_dir on host $HOME"
       /bin/rm -r $HOME/$remote_dir              remove temporary directory
       /bin/rm itough2.fil
       /bin/rm itough2.ver
     else                    parent host and child host are not identical; use  rsh
       echo "PVM: Removing temporary directory $remote_dir on host $host"
       rsh -n $host rm -r $remote_dir      remove temporary directory on host
       rsh -n $host rm itough2.fil
       rsh -n $host rm itough2.ver
     fi
  done
  fi
  cd ..
  /bin/rm -r $tmp_dir
else
  echo "Temporary directory $tmp_dir not removed!"
fi
echo
echo ++++++++++++++++++++
echo + iTOUGH2 terminated: $arguments
echo ++++++++++++++++++++
#
# end of itough2 script file
```

Figure A.1. (cont.) Excerpt from shell script file `itough2`.

APPENDIX B: iTOUGH2-PVM ARCHITECTURE

Figure B.1 shows a simplified flow chart of iTOUGH2-PVM. The source codes for the parent and child processes are identical. All PVM-related subroutines can be found in file `pvm.f`.

iTOUGH2-PVM first enrolls itself into PVM, obtains its task identifier (TID), and determines whether it is a parent process (`IPVMMS=1`) or a child process spawned by a parent process (`IPVMMS=2`). The program flow differs depending on whether the task is a parent or child process.

The parent process reads the standard TOUGH2 and iTOUGH2 input files from the temporary directory `$HOME/it2_<pid>`. The information provided in block `>>> PVM` (see subroutine `PVMINPUT`) is used to add hosts to the virtual machine and to spawn child processes (see subroutine `PVMINIT`). Furthermore, it sends the name of temporary directory `it2_*HOSTiPVM` to the corresponding child process. As soon as the directory name is received by the child process, it starts reading the TOUGH2 and iTOUGH2 input files, which were copied to the host's temporary directory by the shell script `itough2` (see Section 4.2 and Appendix A). After reading of input is completed, the child process goes to subroutine `FCNLEV` and waits for the arrival of data or parameter sets sent by the parent process (see subroutine `PVMRCVPAR`). In the meantime, the parent process performs the initial forward run with the base-case parameter set (except for grid search). If sensitivity analyses, first-order-second-moment (FOSM) uncertainty propagation analyses, or Monte Carlo simulations are performed, the results from the initial run are broadcast to all hosts. No such step is required when performing optimization using the Levenberg-Marquardt, Gauss-Newton, Downhill Simplex, or Grid Search method. After the initial run, the parameter set is updated according to the procedure of the selected method. The updated parameter set is sent to one of the child processes (see subroutine `PVMSENPAR`). The procedure is repeated until all child processes are busy. The child processes perform one TOUGH2 forward calculation with the parameter set they have received from the parent process. After completion of the run, they send the resulting residuals to the parent process (see subroutine `PVMSENRES`) and go back to the top of subroutine `FCNLEV`, waiting for the next parameter set to arrive. The parent process checks for incoming residual vectors (see subroutine `PVMRCVRES`), and processes them according to the selected method. If convergence is achieved or one of the child process signals that it was stopped (see subroutine `PVMRCVRES`), the parent process stops all child processes before it continues with the error analysis in subroutine `TERMINAT`.

If PVM is not engaged, variable `IPVMMS` is set to zero, and iTOUGH2 runs in its standard mode, skipping PVM-related steps. Looking for variable `IPVMMS` in files `it2main.f` and `it2xxxx.f` leads to the code affected by PVM.

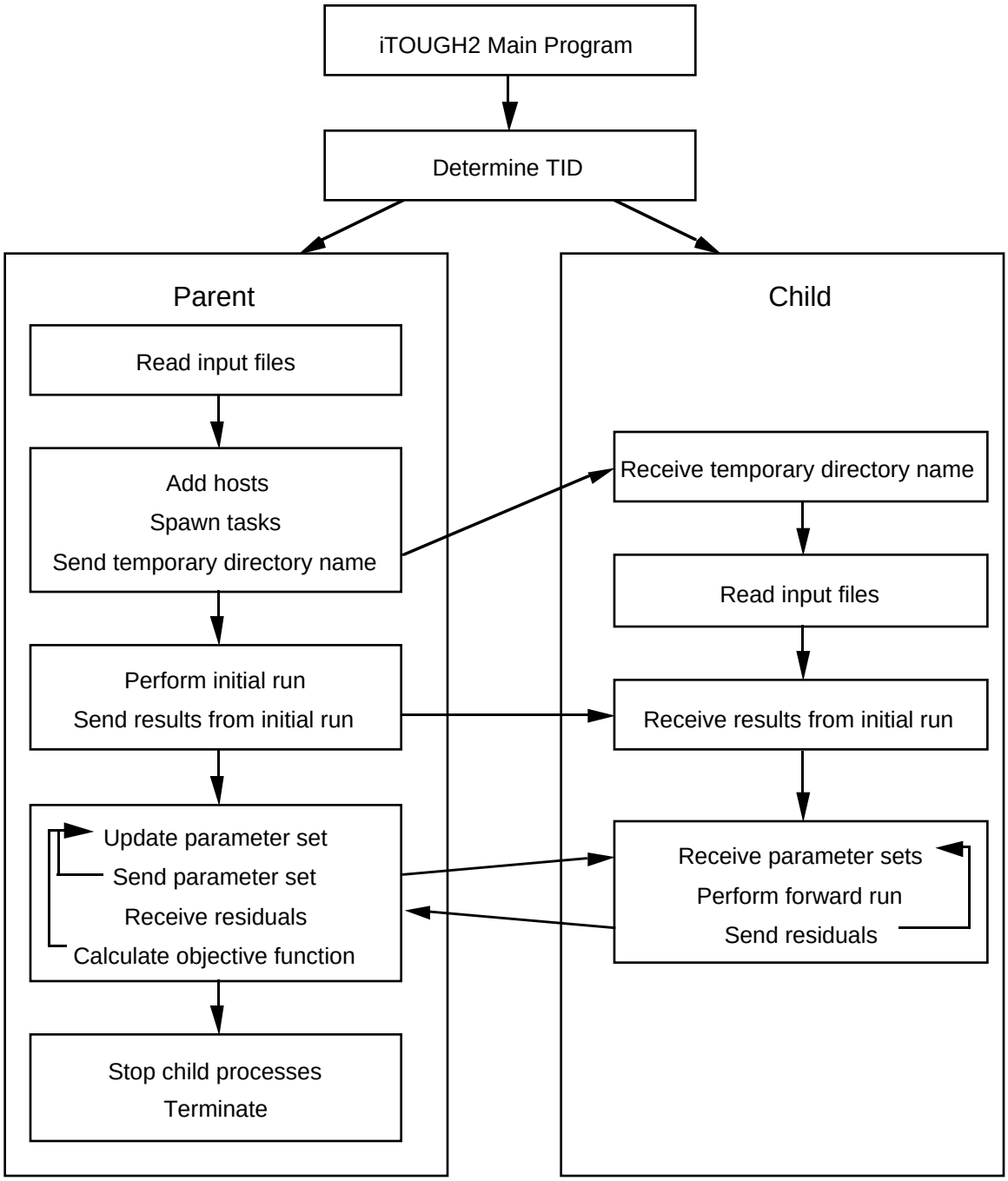

Figure B.1. Simplified iTOUGH2-PVM flow chart.